Concepts and Techniques for Web-based Visualization and Processing of Massive 3D Point Clouds with Semantics

Sören Discher
Hasso Plattner Institute
University of Potsdam, Germany
soeren.discher@hpi.de

Rico Richter
Hasso Plattner Institute
University of Potsdam, Germany
rico.richter@hpi.de

Jürgen Döllner
Hasso Plattner Institute
University of Potsdam, Germany
juergen.doellner@hpi.de

ABSTRACT

3D point cloud technology facilitates the automated and highly detailed acquisition of real-world environments such as assets, sites, and countries. We present a web-based system for the interactive exploration and inspection of arbitrary large 3D point clouds. Our approach is able to render 3D point clouds with billions of points using spatial data structures and level-of-detail representations. Point-based rendering techniques and post-processing effects are provided to enable task-specific and data-specific filtering, e.g., based on semantics. A set of interaction techniques allows users to collaboratively work with the data (e.g., measuring distances and annotating). Additional value is provided by the system's ability to display additional, context-providing geodata alongside 3D point clouds and to integrate processing and analysis operations. We have evaluated the presented techniques and in case studies and with different data sets from aerial, mobile, and terrestrial acquisition with up to 120 billion points to show their practicality and feasibility.

CCS CONCEPTS

• **Human-centered computing** → *Geographic visualization*; • **Computing methodologies** → *Computer graphics*; *Point-based models*.

KEYWORDS

3D Point Clouds, web-based rendering, point-based rendering; processing strategies

ACM Reference Format:
Sören Discher, Rico Richter, and Jürgen Döllner. 2018. Concepts and Techniques for Web-based Visualization and Processing of Massive 3D Point Clouds with Semantics. In *Web3D 2018 Special Issue of Graphical Models*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3208806.3208816

1 MOTIVATION

3D point clouds allow for a discrete representation of real-world objects and environments. They can be time-efficiently and cost-efficiently generated by a large number of acquisition techniques using active or passive sensing technology such as LiDAR, radar, or aerial and digital cameras [Eitel et al. 2016; Ostrowski et al. 2014]. Integrated into a variety of carrier platforms such as airplanes, helicopters, UAVs, cars, trains, and robots, the sensing technology can capture data at different scales, ranging from small assets over buildings and infrastructure networks up to entire cities and countries [Kersten et al. 2016; Langner et al. 2016; Remondino et al. 2013]. The resulting data sets are essential for a growing number of applications in domains such as land surveying, urban planning, landscape architecture, environmental monitoring, disaster management, construction as well as spatial analysis and simulation [Eitel et al. 2016; Nebiker et al. 2010; Pătrăucean et al. 2015].

By their very nature, 3D point clouds are unstructured and do not contain or imply any order or connectivity between individual points. As a consequence, traditional analysis algorithms for geodata often struggle with 3D point clouds as they commonly rely on explicitly defined connectivity information. Visualization algorithms often apply a uniform pixel size and render style to each point and, therefore, are prone to visual artifacts such as holes or visual clutter which severely limits perception, interaction, and navigation [Richter et al. 2015]. As a remedy, GIS applications frequently use 3D point clouds only as input data to derive mesh based 3D models (e.g., 3D city models, terrain models) [Berger et al. 2014]. Depending on the use case, this may require a time-consuming and only semi-automatic process that does not scale for massive data sets, especially if precision, density and data quality of the derived 3D models need to be maximized. Moreover, improved scanning hardware and novel carrier systems, which get cheaper and easier-to-use, result in more dense 3D point clouds. Thus, there is a strong demand to store, manage, process and explore massive, arbitrarily dense 3D point clouds in order to take advantage of their full potential and to provide an unfiltered, detailed representation of captured sites.

In this paper, we present a web-based visualization system for massive 3D point clouds (Fig. 1) based on spatial data structures and level-of-detail representations that provide efficient access to arbitrary subsets of the 3D point cloud stored on a central server component. By combining out-of-core rendering concepts with web-based rendering concepts massive data sets can be simultaneously distributed to and interactively visualized on an arbitrary number of client devices with different computation capabilities. To facilitate a collaborative inspection and to highlight task-relevant aspects of the data, different point-based rendering and interaction techniques are implemented that can be combined and configured by the user. In addition, the system can take advantage of per-point attributes generated by additional point-cloud analysis services. We evaluate our system with real-world data sets containing up to 120 billion points. Results show that the system is capable to provide a powerful

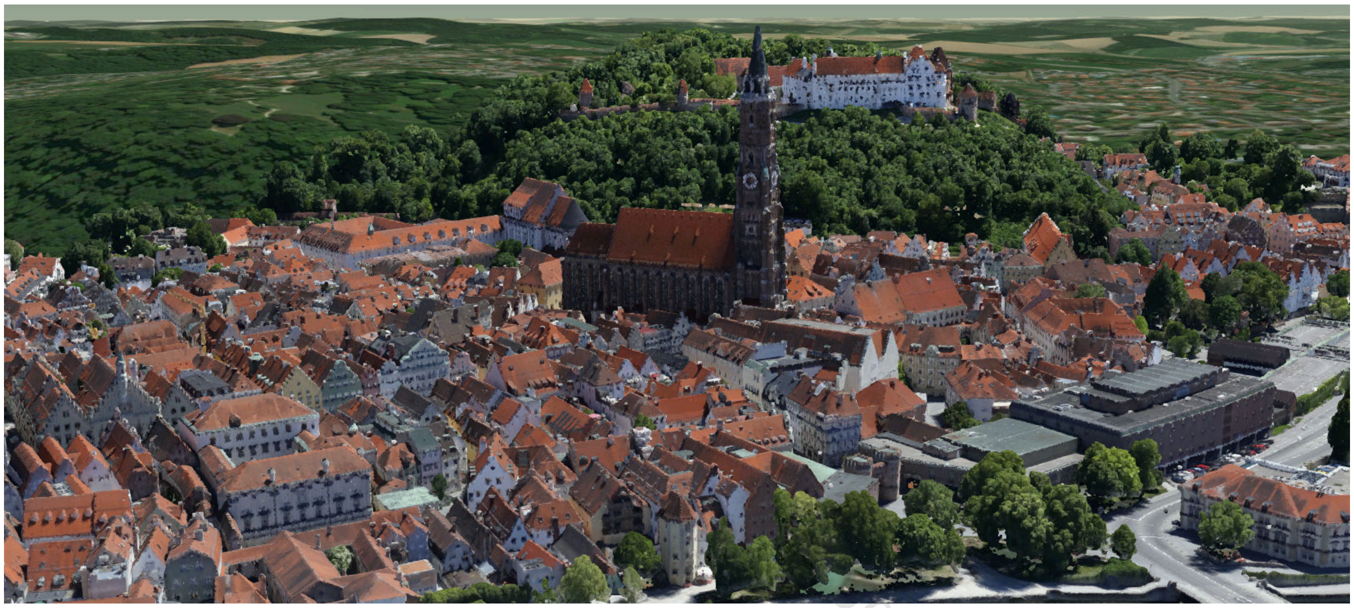


Figure 1: Example of a massive 3D point cloud rendered with our web-based system. Context-providing geodata such as 2D maps and 3D terrain models can be integrated into the visualization.

component in production workflows to manage, distribute and share 3D point clouds.

2 RELATED WORK

3D point clouds represent a universal data category for a large number of geospatial applications [Eitel et al. 2016; Rüther et al. 2012]; many approaches exist to enhance information implicitly contained in 3D point clouds by deriving information about surface categories for each point, typically based on local topological analysis [Chen et al. 2017] or by deep learning concepts [Boulch et al. 2017; Huang and You 2016]. [Richter et al. 2013b] show how to efficiently identify changes in multi-temporal data sets, which contain data acquired at different points in time. [Awrangjeb et al. 2015] combine surface categories and change detection results to filter detected changes based on semantics. Our approach provides efficient means to integrate such analyses as separate web processing services [Müller and Pross 2015]. Furthermore, analysis results can be shared, explored and inspected.

A general overview of point-based rendering techniques is provided by [Gross and Pfister 2011]. High-quality rendering techniques [Preiner et al. 2012; Schütz and Wimmer 2015a] focus on minimizing artifacts such as visual clutter or visible holes between neighboring points by applying appropriate size, orientation, textures and color schemes to each rendered primitive. Often, such techniques apply surface splatting [Botsch et al. 2005], rendering not the points themselves but rather a set of arbitrary-shaped surface patches that approximate the underlying surface of a 3D point cloud. While surface splatting typically requires some preprocessing (e.g., to consolidate redundant surface patches or to apply texturing [García et al. 2015]), it can significantly increase the realism of a point cloud depiction. Non-photorealistic rendering techniques [Simons et al. 2014; Zhang et al. 2014], on the other hand, deal with the fuzziness of a 3D point cloud and highlight edges and structures, commonly without requiring any preprocessing or additional attributes apart from a point's spatial position [Boucheny 2009; Pintus et al. 2011]. In our approach, we implement high-quality as well as non-photorealisitic rendering techniques. They can be switched and configured at runtime.

Out-of-core rendering concepts for massive 3D point clouds were initially introduced by [Rusinkiewicz and Levoy 2000]. State-of-the-art techniques typically use spatial data structures such as layered, regular grids [Deibe et al. 2019], quadtrees [Gao et al. 2014], octrees [Elseberg et al. 2013; Wand et al. 2008], or KD-trees [Goswami et al. 2013] to subdivide the data into smaller subsets that can be selected dynamically, e.g., based on the current view frustum and memory budget.

Recent approaches combine out-of-core and web-based rendering concepts to enable a ubiquitous visualization of 3D point clouds [Rodriguez et al. 2012]. With *Potree*, [Schütz and Wimmer 2015b] propose a thick client approach for arbitrary large data sets, which is adapted by [Martinez-Rubi et al. 2015] to interactively present a massive data set of the Netherlands. An alternative thick client renderer for 3D point clouds named *Plasio* was introduced by [Butler et al. 2014]: Using open-source libraries such as Entwine and Greyhound massive data sets can be streamed interactively. *GVLiDAR* [Deibe et al. 2017] and *ViLMA* [Deibe et al. 2019] constitute thick client rendering approaches that focus on geospatial analysis and measurement tools. While all four frameworks provide effective interaction and inspection techniques specifically for 3D point clouds, they offer only minimal support to integrate additional, context-providing geodata (e.g., shapes, 2D maps). The *Cesium* [1]

[1]https://cesiumjs.org

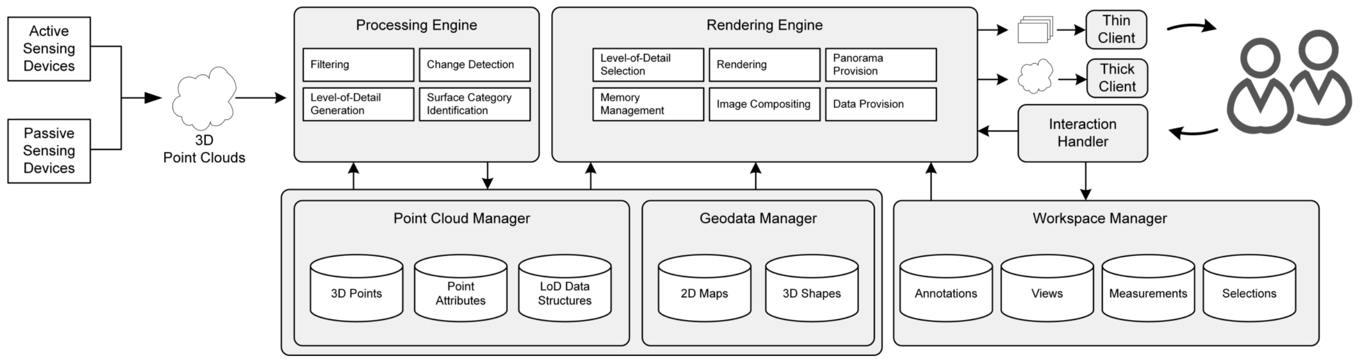


Figure 2: System architecture showing data flow between integration, processing, visualization, and interaction components.

framework on the other hand aims to provide a generalized thick client rendering solution for arbitrary types of geodata (e.g., 3D point clouds, 3D meshes, 2D maps). Our approach is in parts based on that framework, expanding it by several semantics-dependent rendering techniques and a set of interaction techniques for the collaborative inspection of 3D point clouds (e.g., to share, query, and annotate).

In addition, our approach also provides a thin client renderer, allowing to optionally reduce the performance impact on client-side by delegating the rendering to the server side. Compared to the aforementioned frameworks, we can thus adapt to a broader range of computing and graphics capabilities on client side. Similar approaches have been successfully implemented in the past [Christen and Nebiker 2015; Döllner et al. 2012; Gutbell et al. 2016] but typically focus on mesh-based geometry rather than 3D point clouds. To generate stereoscopic panoramas we implemented the theoretical concepts described by [Peleg et al. 2001] by means of modern 3D computer graphics.

Systems for the efficient management of massive 3D point clouds have been recently presented and evaluated by [Cura et al. 2017], [van Oosterom et al. 2017], and [Poux et al. 2016]. However, those contributions focus on the efficient storage, retrieval and processing of the stored data sets. Less emphasis is put on the collaborative exploration, inspection and manipulation of the stored data sets.

3 REQUIREMENTS

We have identified the following requirements that need to be addressed by a system for the web-based visualization and collaborative exploration of massive 3D point clouds:

- **R1** Use of 3D point clouds as a fundamental geometry type instead of generalized mesh-based representations to enable a direct and unfiltered provision of the data.
- **R2** No limitations regarding used acquisition methods as well as density, resolution, and scale of the data (e.g., hundreds of billions of points, complete countries).
- **R3** Support for varying hardware platforms and computation capabilities, ranging from high-end desktop computers to low-end mobile devices.
- **R4** Distributed data storage to enable load balancing and to adjust for data specific requirements (e.g., certain 3D point clouds might have to be stored on a specific server).
- **R5** Capabilities to prepare and clean up 3D point clouds for the visualization (e.g., noise and outlier removal).
- **R6** Capabilities to conduct task and data specific analyses on 3D point clouds (e.g., surface category extraction) to provide adaptive and task specific content.
- **R7** Visualization of analysis results (e.g., surface categories) to enable task specific highlighting and filtering.
- **R8** Capabilities to compare and show differences between 3D point clouds from different points in time of the same site (i.e., change detection).
- **R9** Integration of supplementary, context-providing geodata such as 2D maps.
- **R10** Provision of interaction techniques to inspect (e.g., measuring of distances, areas, volumes) and annotate 3D point clouds.
- **R11** Basic user management to customize data access.
- **R12** Capabilities to share specific rendering configurations, annotations and measurements with others (e.g., via link).

4 CONCEPTS

We have addressed the aforementioned requirements in the design and implementation of our web-based system that seamlessly combines functionality to integrate, process, and collaboratively explore massive, heterogeneous 3D point clouds as well as supplementary, context-providing geodata. The proposed system (Fig. 2) consists of the following major components:

4.1 Point Cloud Manager

In our approach, 3D point clouds are organized in a single, homogeneous spatial data model. Access to that model is handled by the *point cloud manager* storing spatial information and additionally provided or computed per-point attributes (e.g., temporal information or surface categories) **(R1)**. Level-of-detail representations [Elseberg et al. 2013; Goswami et al. 2013] are required to efficiently access arbitrary data subsets of any size based on spatial, temporal or any other attributes. These representations as well as additional per-point attributes can be generated by the *processing engine* (Section 4.4) **(R2)**. While the point cloud manager logically acts as a singular component, the data itself may be stored in a distributed infrastructure, e.g., to maximize data throughput and network transfer rates or to account for data specific requirements regarding server location and data security **(R4)**.

4.2 Workspace Manager

The *workspace manager* handles information specific to a workspace, i.e., each user‘s private view of a specific data subset containing custom selections, measurements, annotations, view positions, and angles. Per default, each user operates in its own private workspace rather than sharing one globally with everyone else to avoid conflicting modifications **(R11)**. However, a given workspace may be shared via links **(R12)**. Each user may also own multiple workspaces.

4.3 Geodata Manager

By application-specific geodata, we refer to additional geodata that should be used and rendered in combination with a 3D point cloud to provide application-specific information layers **(R9)**. Examples are digital terrain models, aerial images, BIM models, or 3D city models. Similar to 3D point clouds, these data types also require supplemental level-of-detail representations to allow for an interactive visualization. Application-specific geodata can be stored and provided by independent geospatial databases or geodata services, access to which is handled by the *geodata manager*.

4.4 Processing Engine

The processing engine conducts task and data specific operations on a given data subset. These operations range from (a) essential preprocessing steps (e.g., converting input data sets into a homogeneous georeference system or generating level-of-detail representations), over (b) simple point cloud filtering (e.g., noise and outlier removal **(R5)**) to (c) more complex analyses (e.g., surface category extraction and change detection **(R6)**) deriving additional per-point attributes. The operations can be accessed via web processing services implemented as separate web services that are individually combined and scheduled by the processing engine. Thus, existing web processing services for 3D point clouds can be easily integrated into the system. The results of each operation are automatically stored by the point cloud manager and can be seamlessly integrated by the *rendering engine* (Section 4.5) into depictions of the corresponding site **(R7)**.

4.5 Rendering Engine

Providing the core functionality of our system, the rendering engine is responsible for interactively visualizing three types of data: (a) 3D point clouds featuring a varying number of per-point attributes, (b) task-specific geodata providing context (e.g., maps **(R9)**), and (c) workspace elements resulting from user interactions (e.g., annotations or selection and measurement indicators **(R10)**). For each of those data types the corresponding manager is queried, retrieving only data subsets that are relevant for the current view and task. To highlight certain aspects of the data (e.g., temporal changes or surface categories in an area), different point-based rendering techniques and post processing effects can be combined **(R8)**. Changes to the currently applied render configuration can be made dynamically via the *interaction handler*) (Fig. 4). In general, retrieved data subsets will be transferred to and rendered on client side, which minimizes the workload on the server (i.e., thick clients). As an alternative, server-side rendering can be applied to reduce the performance impact for clients (i.e., thin clients). Thus, the system scales for a broad range of devices, ranging from high-end workstations to mobile devices **(R3)**.

4.6 Interaction Handler

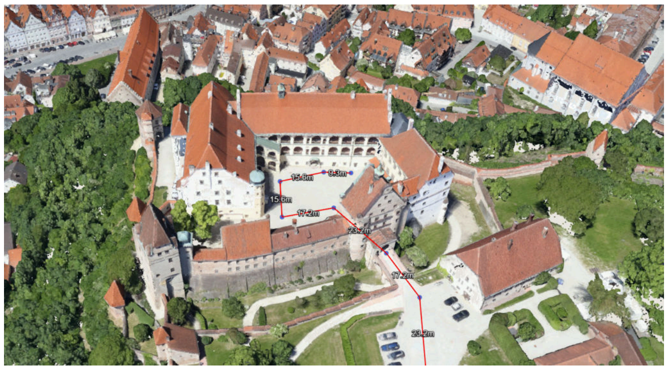
The interaction handler is responsible for handling user interactions as well as for updating the rendered data and workspace elements accordingly **(R10)**. Users may

- define or load workspaces,
- select which data subsets to render,
- configure the presentation of the data (with regards to applied rendering techniques),
- select, query and highlight individual points or groups of points,
- measure distances and areas between selected points,
- annotate selected points or areas,
- modify annotations,
- saving and loading view positions and angles.

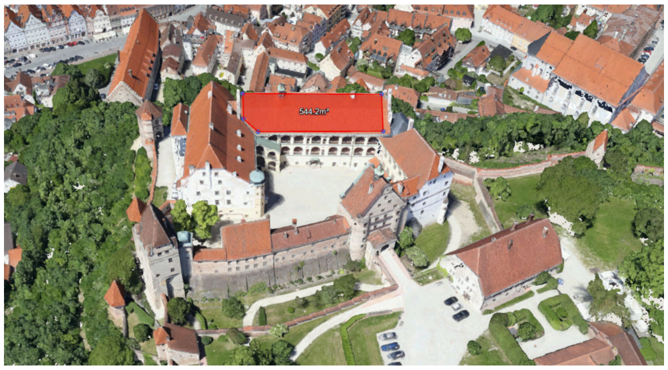
5 PROCESSING ENGINE IMPLEMENTATION

Data and use case specific operations on 3D point clouds typically combine several atomic processing steps (e.g., determining a point's closest neighbor or aggregating attribute values within the local neighborhood of a point) that can be executed independently on a small area around each point [Boulch et al. 2017; Chen et al. 2017; Richter et al. 2013a,b]. Therefore, the processing performance can be significantly increased by applying parallel computing concepts, either based on a CPU or a GPU. Different processing and analysis operations can be efficiently chained together by interleaving them. Instead of executing each operation one at a time for the complete data set, processed subsets are immediately subjected to subsequent operations. Since 3D point clouds commonly exceed available capacities of main or GPU memory, these parallel computing concepts need to combined with out-of-core approaches that subdivide the overall data set into sufficiently small subsets.

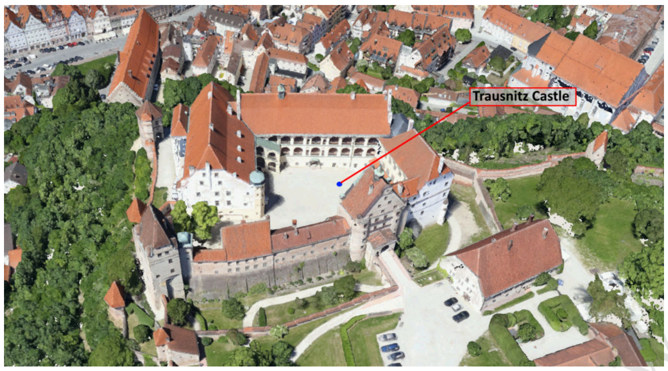
The processing engine uses a modular *pipeline architecture* (Figure 5) that combines parallel computing and out-of-core concepts: Basic processing and analysis operations can be freely combined and applied to arbitrary large data sets, making optimal use of available hardware resources by parallelizing, interleaving, and distributing operations alongside corresponding data subsets between computing resources (e.g., different servers in a distributed environment). The processing and analysis operations involved are described via processing pipelines that can be reconfigured and replaced at runtime. We allow for an efficient retrieval of arbitrary subsets by means of a multi-layer hierarchical subdivision: For each 3D point cloud, a separate spatial data structure is generated that best compliments the spatial distribution of the corresponding points (e.g., quadtrees for airborne data sets, octrees or kd-trees for terrestrial data sets). In turn, those spatial data structures are integrated into an overarching quadtree, allowing to efficiently answer queries stretching across multiple data sets. Compared to uniform, single-layer spatial data structures (e.g., as they are used by [Schütz and Wimmer 2015b]), this avoids a time consuming rebalancing when new 3D point clouds are added to the system while



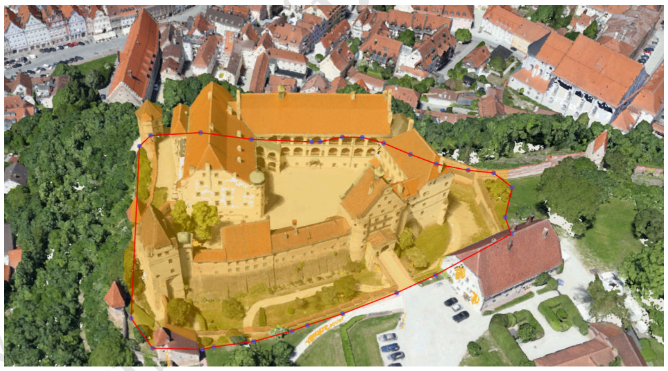
(a) Measuring of distances between points.



(b) Measuring of areas defined by multiple points.



(c) Annotation of selected points or areas.



(d) Selecting areas of interest.

Figure 3: Overview of implemented interaction techniques for 3D point clouds.

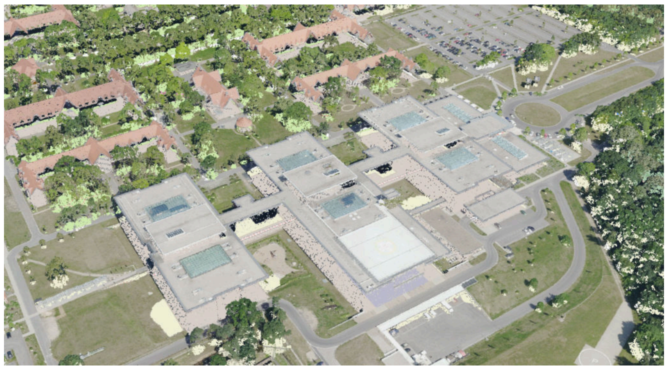
simultaneously ensuring balanced tree structures and minimal data access times. Context providing geodata and workspace elements are organized in similar fashion.

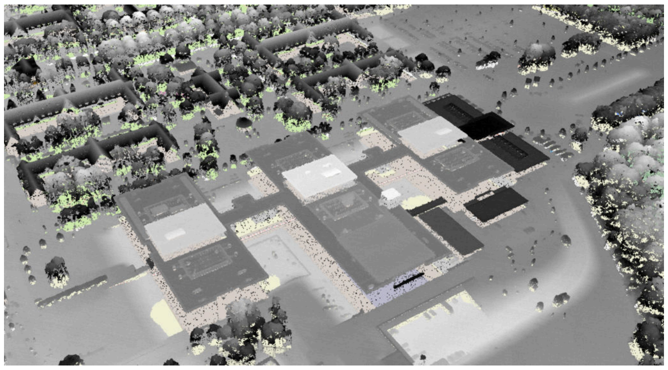
5.1 Pipeline Architecture

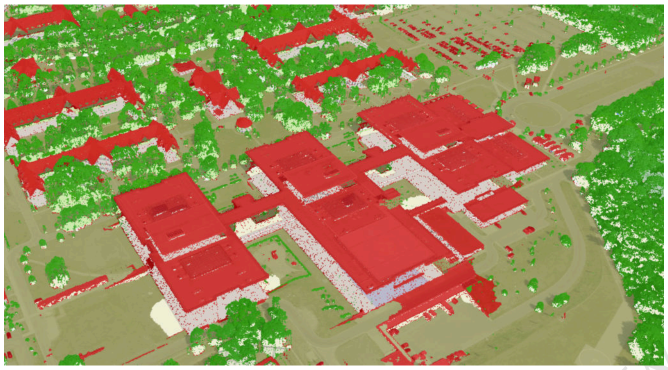
The applied pipeline architecture is based on the concept of so-called *processing pipelines*, each of which described by a pipeline plan defining a specific combination of basic input, processing and output operations. To be more precise, a pipeline plan may contain the following elements (also referred to as *pipeline nodes*):

- **Importers**, i.e., pipeline nodes that import 3D point clouds, either file based or directly from the point cloud manager. For each data source and file format (e.g., LAS, E57) a separate importer is provided. Each importer prepares data packages. If the input data exceeds the maximum data package size, the importer prepares subsets by splitting the data.
- **Exporters**, i.e., pipeline nodes that export pipeline results. This can refer to Level-of-Detail representations, additional point attributes or artificially generated 3D point clouds, but also to additional geodata (e.g., shape files, CityGML, GeoTIFFs). All results are then stored by the corresponding manager, allowing to seamlessly integrate them into the visualization.
- **Tasks**, i.e., pipeline nodes that implement a specific processing or analysis algorithm. Some algorithms operate on multiple data sets simultaneously (e.g., to compare or to merge them). Similarly, algorithms may split incoming data sets or yield multiple results (e.g., additional per-point attributes and corresponding shapes). Hence, multiple incoming and outgoing connections may be defined per task.
- **Connections**, i.e., links between two pipeline nodes for the transfer of data packages. They define the order of execution. A given connection transfers only packages of a specific type (e. g., 3D point clouds or shapes). Depending on the pipeline nodes being connected, various constraints may be defined, such as defined per-point attributes that are required.
- **Data Packages**, i.e., data subsets that are transferred between pipeline nodes via connections. Similar to connections, a given data package may only contain a specific type of geodata. Also, the size of the corresponding data subset may not exceed a specific maximum defined by the resource manager.

Pipeline plans are executed by the *pipeline engine*. Several pipeline plans can be executed in parallel; every single one can also be dynamically started, paused and stopped. At runtime, each active pipeline node gets assigned its own set of resources by the *resource*


(a) Colors extracted from aerial imagery.


(b) Height-based gradient from black to white.


(c) Colorization based on surface categories, i.e., green for vegetation, red for buildings, and brown for ground.

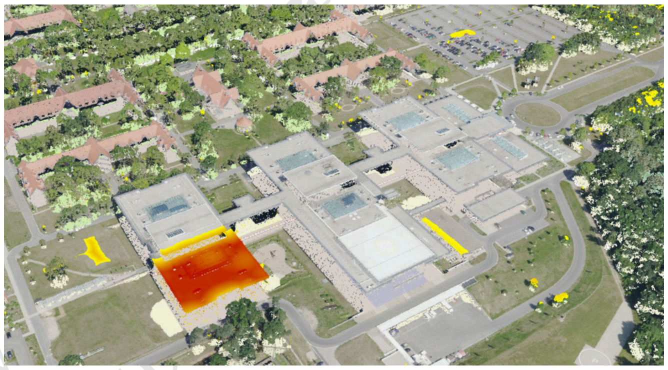

(d) Colors from aerial imagery combined with change detection results, i.e., gradient from yellow to red indicates the degree of change.

Figure 4: Different point-based rendering styles can be selected and configured at runtime.

manager, responsible for monitoring and distributing memory and processing usage within a system. Processed data packages are immediately transferred to subsequent pipeline nodes. Pipeline nodes manage a queue of incoming data packages for each incoming connection, whose size is restricted to a maximum number of data packages that can be defined at runtime. If a queue reaches its maximum capacity, no additional data packages are accepted and preceding nodes are not executed. To improve their runtime performance, the most time-consuming pipeline nodes are executed in parallel by adaptively assigning additional resources (e.g., CPU or GPU cores).

5.2 Memory and Resource Management

The resources of a system may be distributed across several servers in a network, each featuring different memory capacities (i.e., size of secondary storage, main memory, and GPU memory) and computing capabilities (e.g., number and clock speed of CPU and GPU cores, memory transfer rates). Servers and their resources are added to a global *resource pool* that is monitored by the resource manager of the system. Whenever a pipeline node needs to be executed, the resource manager assigns resources based on available memory and processing capabilities. After the execution is finished, all assigned resources are released to the resource pool and become available for other nodes (Figure 5). Distributing resources requires the resource manager to make a trade-off between several, often contradicting optimization goals:

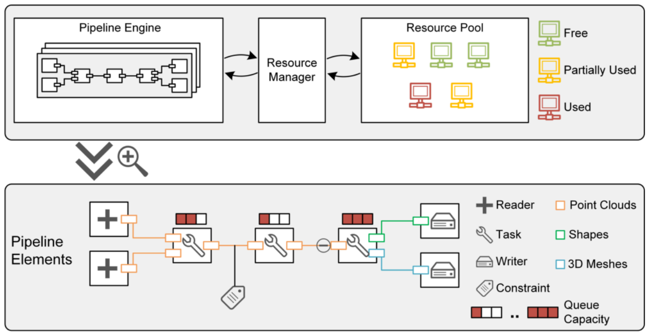


Figure 5: Overview of the pipeline architecture and pipeline elements.

- **Exclusivity.** Exclusive access to a resource (e.g., storage or GPU) significantly improves the runtime performance of a pipeline node (e.g., by minimizing cache misses and seek times).

- **Transfer Costs.** Frequently transferring data packages via connections may notably reduce the performance if subsequent pipeline nodes operate on different servers. This can be avoided by executing them on the same server.
- **Parallelization.** Executing pipeline nodes in parallel or interleaved is an essential mechanism to improve the overall performance of the system. Thus, available resources and servers should be shared among as many pipeline nodes as possible.

The runtime of nodes may vary significantly depending on the operation. An adaptive resource scheduling allows to handle bottlenecks in processing pipeline. The execution time is tracked for each node and the number of assigned resources is adjusted dynamically.

6 RENDERING ENGINE IMPLEMENTATION

To seamlessly combine 3D point clouds, context providing geodata and interactive workspace elements into a homogeneous visualization, a multi-pass rendering pipeline is used that consists of three distinct stages (Fig. 6):

6.1 Level-of-Detail and Data Subset Selection

While 3D point clouds may easily contain billions of points, only a fraction of that data is required to render a frame. Subsets of the 3D point cloud that are manageable by available CPU and GPU capabilities can be queried dynamically from the point cloud manager by specifying the current view frustum, main and GPU memory budgets as well as task specific qualifiers (e.g., value ranges for selected per-point attributes) to filter the corresponding data sets. In particular, we use the resulting *screen-space error* as a metric to evaluate potentially fitting subsets, optimizing towards a *maximum allowed screen-space error*: The higher the allowed screen-space error, the less points need to be queried and rendered, albeit at the cost of further reducing precision and density of the point cloud depiction. To accommodate for changing network latencies, the maximum can be adjusted at runtime. Since hardware specifications of clients can not be queried from a web browser, main and GPU memory budgets need to be specified manually. To assist users in specifying a reasonable budget, we optionally provide a set of predefined budget configurations that have been tested for a variety of common hardware setups. To enable an efficient subset retrieval, the data is hierarchically subdivided in a pre-processing step (Section 5) before being made accessible by the point cloud manager. Context providing geodata and workspace elements are handled similarly and can be queried simultaneously from their respective manager when required.

6.2 Rendering

After being queried from the respective managers, 3D point clouds, context providing geodata and interactive workspace elements are rendered into separate g-buffers [Saito and Takahashi 1990], i.e., specialized frame buffer objects (FBO) combining multiple 2D textures for, e.g., color, depth, normal, or id values. The use of id values is important to separate point clouds from context data. Each rendered point has a unique identifier, stored in an id texture, to allow for an efficient point selection, e.g., to implement interaction features (Section 6.3). In addition, different rendering styles can

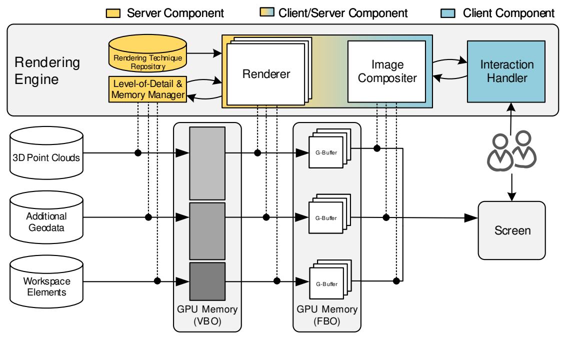


Figure 6: Overview of the rendering pipeline. Each data type is managed and rendered separately.

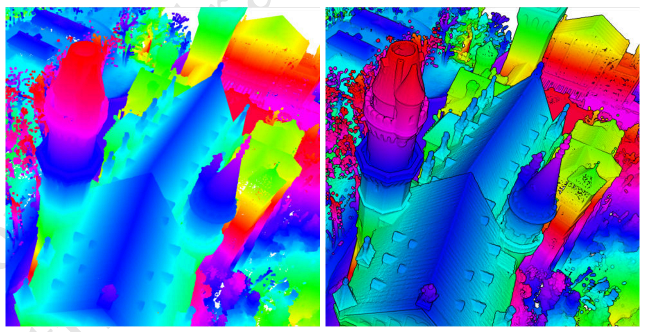


Figure 7: Post-processing effects such as Eye Dome Lighting facilitate visual filtering and highlighting.

be configured and applied at runtime. As an example, size and color of each point can be modified based on selected per-point attributes (e.g., surface categories, topological metrics) to enable task specific visual filtering and highlighting (Fig. 4). Similarly, several options exist to dynamically adjust the appearance of mesh-based geometry, ranging from transparency settings to changeable texture mappings.

6.3 Image Compositing

A final image compositing stage is used to merge the separate g-buffers, i.e., to combine several independently generated views of 3D point sub-clouds into a final image. For example, image-based post processing effects emphasizing edges and depth differences (e.g., Screen Space Ambient Occlusion [Mittring 2007] or Eye Dome Lighting [Boucheny 2009]) can be applied at that stage to improve the visual identification of structures within 3D point cloud depictions (Fig. 7). The id textures stored by the g-buffers provide efficient means to identify which point was rendered at a specific pixel. Thus, individual points can be selected in real-time, which is an essential requirement to support annotating points or measuring distances and areas.

Table 1: Devices used to evaluate the rendering engine. All web browsers were updated to the latest version as of 04/20/2018.

Client Device	CPU	Main Memory	GPU	Evaluated Web Browsers
Lenovo M710t	Intel Core i7-6700	32GB	GeForce GTX 1050Ti	Chrome, Firefox, Opera, Edge
Macbook Pro 13”	Intel Core i5-4278U	16GB	Intel Iris 5100	Safari, Chrome, Firefox
iPhone SE	Apple A9 @ 1.84 GHz	2GB	PowerVR GT7600	Safari Mobile, Chrome Mobile
Galaxy s7	Samsung Exynos 8890	4GB	ARM Mali-T880 MP12	Samsung Internet Browser, Chrome Mobile

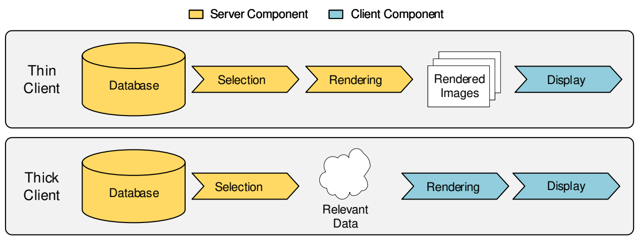


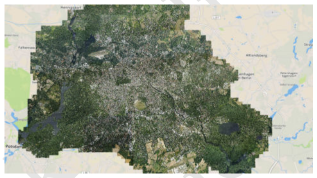
Figure 8: Comparison of web-based rendering concepts: Thin clients vs thick clients

6.4 Web-based Rendering

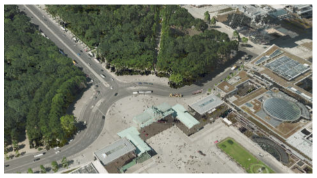
To accommodate for client devices with varying computation capabilities, different web-based rendering concepts are combined with the presented rendering pipeline (Fig. 8). We provide a thick client application that uses a central server infrastructure to organize, process, select and distribute the data, but delegates the actual rendering of selected data subsets to the clients. This approach significantly reduces workload on server side, allowing to serve massive numbers of clients simultaneously. Transferred data subsets are cached on client side up to a device specific limit, thus, minimizing the frequency of data requests for subsequent frames. In fact, additional data subsets are only required if the view frustum changes significantly, whereas inspecting the transferred data or changing the applied rendering style triggers no such requests. Alternatively, in the sense of a thin client approach, the data can be rendered directly on the server, supplying only the resulting images. While this comes with the drawback of increased workload on server side as any user interactions trigger a new data request, hardware requirements for clients are notably reduced. A common optimization for such thin client applications is to render and transfer cube-maps or virtual panoramas instead of individual images [Döllner et al. 2012; Hagedorn et al. 2017]. This provides clients with efficient means to locally reconstruct the 3D scene for a specific view position. Thus, the data only has to be rendered anew whenever the view center or the rendering style are modified, which significantly reduces the frequency of data requests for subsequent frames. Our system provides a thin client application that expands that concept, distributing not only traditional 2D panoramas but also stereoscopic panoramas. Thus, emerging virtual reality technologies allowing for an immersive exploration of 3D point clouds even on mobile devices can be easily integrated. We generate those stereoscopic panoramas by rendering several equally-sized image strips along a *viewing circle* that are stitched together in a post-processing step [Peleg et al. 2001]. The visual quality of the

(a) Terrestrial indoor scan.

(b) Mobile mapping scan.



(c) Airborne scan of a city (zoomed out)



(d) Airborne scan of a city (zoomed in)

Figure 9: Scenes used during the intial performance tests.

Table 2: Average data throughput of the processing engine.

Processing Operation	Average Data Throughput
Noise & Outlier Filtering	1.26B pts/hour
Surface Category Extraction	0.10B pts/hour
Change Detection	1.42B pts/hour
Kd-Tree Generation	4.85B pts/hour

panoramas depends on the requested resolution as well as the number of image strips; both settings can be specified upon requesting a new panorama. To further reduce overall network load, both applications dynamically compress and decompress the transferred data, using common standards such as gzip (for thick clients) and png (for thin clients), respectively. We decided against using any lossy compression standards (e.g., jpeg compression) to maximize visual quality.

7 EVALUATION

We have implemented the presented concepts on the basis of several C++ and Javascript libraries. The processing engine uses *CUDA* [2] and the *Point Cloud Library* [3]. Regarding the rendering engine, we use WebGL and *Cesium* [4] for thick client applications. For thin client

[2]https://developer.nvidia.com/cuda-zone
[3]http://pointclouds.org
[4]https://cesiumjs.org

applications, server-side rendering is based on OpenGL, *glbinding* [5] and *GLFW* [6]. On client-side, *Three.js* [7], *WebGL*, and *WebVR Polyfill* [8] are combined to display 2D as well as stereoscopic panoramas. For data compression we use *gzip* [9] and *lodePNG* [10], respectively. Evaluated 3D point clouds are represented by separate kd-trees, that in turn are integrated into an overarching quadtree. We opted to use kd-trees to optimize the balancedness of the tree structures, speeding up the subset retrieval, albeit at the cost of a prolonged preprocessing. Those spatial data structures and corresponding data subsets are serialized into files acting as a point cloud database. Similar, file-based approaches are applied to store and access context-providing geodata and workspace elements.

7.1 Performance Tests

A desktop computer featuring an AMD Ryzen 7 1700 CPU, 32 GB main memory and an NVIDIA GeForce GTX 1070 with 8 GB device memory was used as a server for the initial performance tests. The test data sets included a terrestrial, indoor scan of an individual site (1.33 billion points), a mobile mapping scan (2.57 billion points) and a massive, multi-temporal data set of an urban region (120 billion points) captured by airborne devices. For all data sets essential preprocessing steps (i.e., spatial data structure generation) and filtering (i.e., noise and outlier removal) were performed by the processing engine. In addition, surface categories (i.e., ground, building, vegetation) and changes in comparison to earlier scans were extracted for the airborne data set, allowing to evaluate the system's ability to dynamically combine different rendering styles. The average data throughput for the applied processing operations is listed in Table 2.

The rendering engine was evaluated based on four different scenes (Fig. 9) with client applications running on a number of different devices and web browsers (Table 1). As opposed to aforementioned state-of-the-art rendering frameworks for 3D point clouds such as Potree, Plasio, GVLiDAR or ViLMA, our rendering engine provides both, a thick client and a thin client renderer. Thus, the rendering process can be shifted dynamically between client and server side depending on network conditions and a client's computing and graphics capabilities, allowing us to support a broader range of hardware platforms.

Our Cesium-based thick client implementation allows to render several millions of points simultaneously at interactive frame rates (i.e., >30 fps) on standard desktop computers and notebooks (Table 4). On mobile devices, frame rates are significantly lower due to the more limited computing capabilities. However, arbitrary large data sets can be visualized on all evaluated devices by assigning device-specific memory budgets, thus, limiting the density of the point cloud depiction. Overall, rendering performance and visual quality are similar to what can be achieved by aforementioned state-of-the-art approaches such as Potree, Plasio, GVLiDAR or ViLMA.

[5]https://github.com/cginternals/glbinding
[6]http://www.glfw.org
[7]https://threejs.org
[8]https://github.com/immersive-web/webvr-polyfill
[9]http://www.gzip.org
[10]http://lodev.org/lodepng/

However, our thick client implementation allows to seamlessly integrate additional geodata as well as analysis results which greatly facilitates an in-depth inspection.

On the other hand, our thin client implementation provides a uniform rendering quality on all client devices since the panoramas are generated on server side, minimizing workload on client side. On all evaluated devices we measured frame rates close to the corresponding display's refresh rate (e.g., 60 fps on the Galaxy S7), making our approach applicable to state-of-the-art VR devices such as GearVR or Oculus Rift. The performance of the panorama generation is primarily influenced by the requested resolution and to a lesser degree on the number of image strips used (Table 5).

For all evaluated scenes, thick client applications require to transfer significantly more data for an individual scene than thin clients as long as no reusable data subsets have been cached from previous requests, even if gzip compression is applied (Table 3). However, they do not require all those data subsets at once, allowing to update the scene progressively. Furthermore, while exploring a 3D point cloud, the view will usually change only gradually across subsequent frames, allowing for thick clients to reuse many of the previously transferred data subsets, thus, resulting in smaller and faster scene updates over prolonged explorations. Changes to the rendering style as well interaction techniques such as picking, selecting or measuring don't trigger any additional data requests at all and can be applied even under unstable network conditions. For thin client applications on the other hand, no parts of the previously transferred data can be reused if the currently used panorama becomes invalid: Navigating -apart from merely looking around from a fixed position- as well as rendering style adjustments require the server to generate and transfer a new panorama as a replacement. Similar to thick clients however, picking, selecting or measuring can be conducted on the already transferred data and does not trigger any new data requests.

7.2 Case Studies

The initial performance evaluation was followed up by two case studies to demonstrate the scalability of our approach with regards to user base, data size as well as available computing and graphics capacities.

The first case study focused on the collaborative interaction with 3D point clouds in the context of a large-scale infrastructure project involving up to 10 concurrent users representing different stakeholders spread across Germany. The infrastructure project comprised several individual sites that were captured by air or – in the case of some especially relevant building complexes – via terrestrial scanning. In total, the scans amounted to 5.31 TB of raw data (E57 or las format) distributed across 144 individual data sets, each of which containing between 18 million to 4.1 billion points at an average point density of 6.1 points/m^2 (airborne scans) and 1.2 million points/m^2 (terrestrial scans), respectively. Via a web frontend (Fig. 10) users were able to (1) upload data sets asynchronously, (2) georeference them individually and (3) restrict data access to specific users. Simultaneously – given corresponding data access rights – users could collaboratively inspect and annotate 3D point clouds that have already been added to the system as described in Section 4. Rendering performance on client devices was consistent

Table 3: Average data throughput of the rendering engine based on the scenes defined in Fig. 9. For thin clients, a stereoscopic panorama was created per request. While the same, device-dependent resolution was requested for each scene, different entropies affected the compressed image size.

Scene	Thick Client		Thin Client		
	Transferred Data	Transfer Time	Transferred Data	Panorama Generation Time	Transfer Time
Terrestrial	156.2 MB	16.18s	4.68 MB	5.27s	1.36s
Mobile Mapping	140.7 MB	14.15s	4.16 MB	5.05s	1.27s
Airborne (zoomed out)	16.1 MB	3.43s	4.15 MB	4.96s	1.22s
Airborne (zoomed in)	82.4 MB	8.09s	4.54 MB	5.13s	1.32s

Table 4: Average performance rate of the thick client for different point budgets based on the airborne data set (Fig. 9d).

Number of Points	Transferred Data (uncompressed)	Transferred Data (compressed)	Lenovo M710t	Macbook Pro 13"	iPhone SE	Galaxy s7
2M pts	29.5 MB	26.2 MB	122.63fps	53.85fps	41.83fps	39.96fps
4M pts	57.4 MB	50.9 MB	84.48fps	45.63fps	36.44fps	35.29fps
6M pts	85.6 MB	76.1 MB	63.23fps	39.08fps	26.36fps	24.83fps
8M pts	113.6 MB	100.7 MB	56.87fps	35.83fps	19.65fps	18.43fps

Table 5: Panorama generation time for different configurations based on the terrestrial data set (Fig. 9a).

Resolution	Transferred Data	Panorama Generation Time		
		90 image strips	120 image strips	160 image strips
2360x1600 px	2.33 MB	1.88s	2.26s	2.29s
2360x3200 px	4.68 MB	4.20s	4.68s	5.27s
2360x6400 px	9.35 MB	6.17s	7.14s	7.88s

with the results presented in 7.1. A dedicated server featuring an Intel Core i7-8700 CPU, 64 GB main memory and 12 TB secondary storage was used to host uploaded data sets and conduct necessary pre-processing operations. The applied processing pipeline was rather simplistic, combining just three pipeline nodes (Fig. 11): An *importer* and an *exporter*, connected via a *kd-tree generator task*. To speed up performance, each kd-tree generator uses a main memory cache. In the context of this case study, the maximum cache size was set to 16 GB, thus, the number of kd-trees that could be generated in parallel was limited to four in the worst case (Fig. 12). However, even for the largest uploaded data sets pre-processing times stayed below 60 minutes. Furthermore, they were added only gradually which further minimized the delay noticeable by the users.

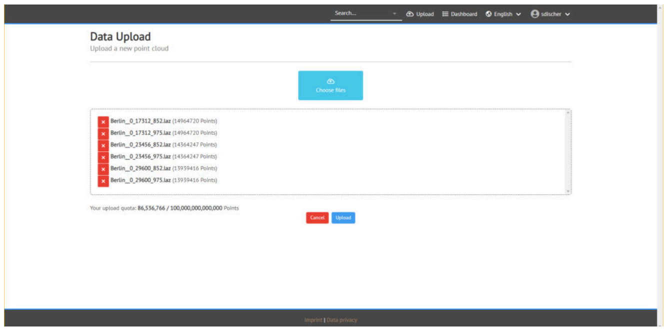
For the second case study, emphasis was put onto the processing engine's performance and scalability in more computation intense scenarios. Hosted on an Oracle Server – featuring an Intel Xeon Gold 5120M CPU, 192 GB main memory, 8 TB secondary storage, and two NVIDIA Tesla P100 with 16 GB device memory – we conducted a change detection as well as a surface category extraction for all uploaded data sets using the processing pipeline depicted in Fig. 11. Users were then able to interactively inspect the processing results by switching between different rendering styles as described in Section 4. Test data for that case study consisted of two different airborne scans of a rural area of 270 km^2 featuring 4 points/m^2 (1.012 billion points in total) and 9 points/m^2 (2.474 billion points in total), respectively. The computation time of both processing operations was notably reduced by increasing the number of CPU threads used in parallel (Fig. 13 and Fig. 14) which underlines the scalability of our approach. The performance of the change detection could be improved further by increasing the number of GPUs used during the computation (Fig. 15), whereas that had only a neglectable effect on the – in that case – mostly CPU based surface category extraction.

7.3 Expandability

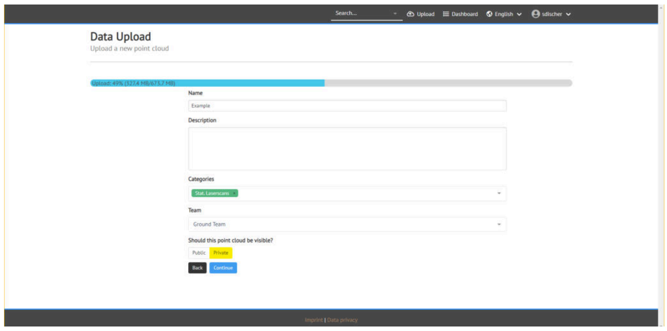
The proposed web-based system can be easily adapted for specific applications by adding custom visualization techniques or algorithms for data analysis: Per-point attributes as well as the pipeline nodes used by the processing engine share common interfaces which facilitates the implementation of additional *importers*, *exporters* or *tasks*. The corresponding *pipeline plans* are defined via JSON files and can thus be easily customized. Similiarly, the rendering engine allows to define and apply custom GLSL shaders to adapt the visualization.

8 CONCLUSION AND FUTURE WORK

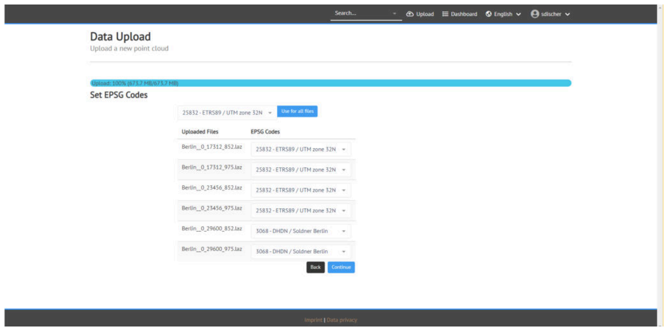
Web-based visualization and exploration of massive 3D point clouds from aerial, mobile, or terrestrial data acquisitions represent a key feature for today's and future systems and applications dealing with digital twins of our physical environment. In our web-based



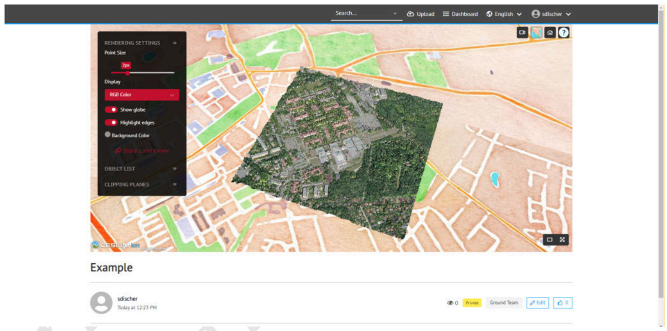
(a) File selection and upload.



(b) Definition of meta data such as data access rights.



(c) Definition of EPSG codes to georeference uploaded data sets.



(d) Visualization of the uploaded and configured data.

Figure 10: Web frontend used during the first case study allowing users to upload, prepare and explore 3D point clouds.

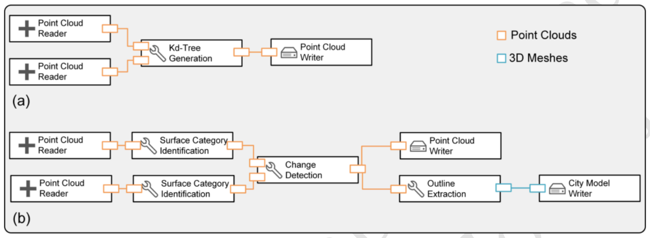


Figure 11: Processing pipelines as they have been used for the presented case studies.

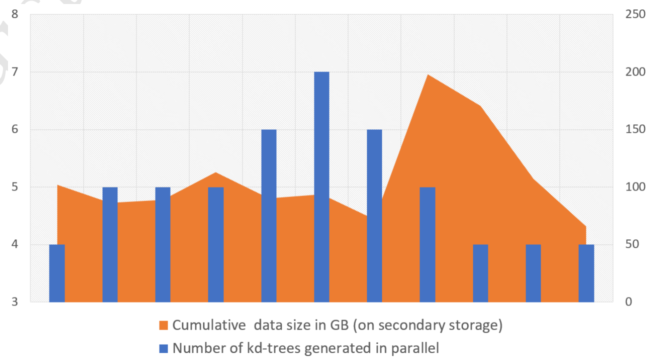


Figure 12: Cache size was limited to 16 GB per kd-tree generator for the case study. Hence, the number of kd-trees that could be generated in parallel (blue) varied depending on the overall size of the corresponding raw data (orange).

approach, we show a system architecture that scalably visualizes massive 3D point clouds to web-based client devices. To cope with extremely large number of points, the implementation relies on spatial data structures and level-of-detail representations, combined with different out-of-core rendering and web-based rendering concepts. Since the rendering process can be shifted from client side to server side, the system can be easily adapted to varying network conditions and to clients with a broad range of computing and graphics capabilities. Tests and case studies on data sets with up to 120 billion points show the usability of the system and the feasibility of the approach. As future work we plan additional case studies with regard to our system's performance in distributed server environments. Various rendering techniques allow us to filter and highlight subsets of the data based on any available per-point attributes (e.g., surface categories or temporal information), which is required to build task-specific or application-specific tools. Various interaction methods (e.g., for collaborative measurements and annotations), built-in support to display context-providing, mesh-based geodata, and the possibility to conduct different processing and analysis operations provide additional features. Our system could be further extended by integrating additional analyses (e.g., for asset detection, or surface segmentation) [Jochem et al. 2012; Teo and Chiu 2015] as well as by specialized interaction techniques. For example, [Scheiblauer and Wimmer 2011] and [Wand et al. 2008] propose

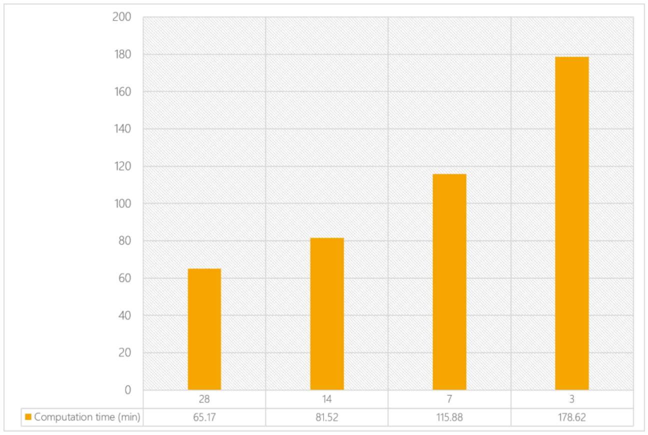


Figure 13: Computation time in minutes of the change detection for different numbers of CPU cores used (for one GPU).

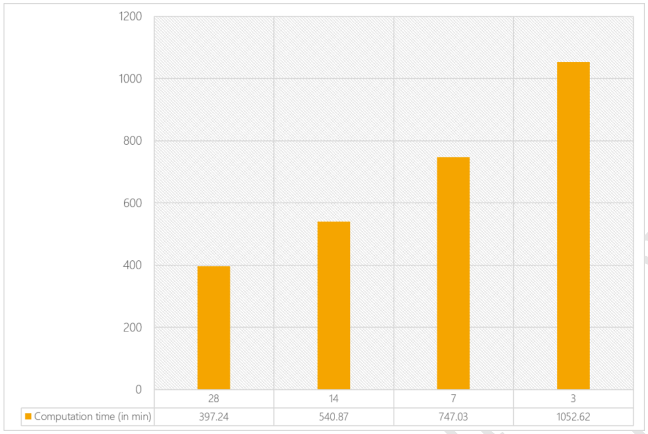


Figure 14: Computation time in minutes of the surface category extraction for different numbers of CPU cores used (for one GPU).

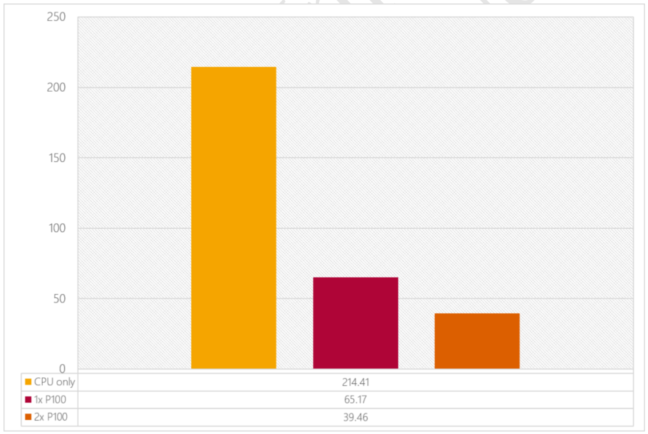


Figure 15: Computation time in minutes of the change detection for different numbers of GPUs used (for 28 CPU cores).

spatial data structures that allow for an interactive editing of 3D point clouds. In addition, sophisticated visualization techniques for multi-temporal 3D point clouds are becoming more and more important to understand captured environments.

ACKNOWLEDGMENTS

We thank Pawel Böning and Pascal Führlich for their contributions to the thin client implementation as well as Heiko Thiel, Marcel Wendler and Pawel Glöckner for their input on the thick client implementation. Presented data sets have been provided by Illustrated Architecture, SHH sp. z o.o. and virtualcitySYSTEMS.

REFERENCES

Mohammad Awrangjeb, Clive S Fraser, and Guojun Lu. 2015. Building change detection from LiDAR point cloud data based on connected component analysis. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 2 (2015), 393–400.

Matthew Berger, Andrea Tagliasacchi, Lee Seversky, Pierre Alliez, Joshua Levine, Andrei Sharf, and Claudio Silva. 2014. State of the art in surface reconstruction from point clouds. In *EUROGRAPHICS star reports*, Vol. 1. 161–185.

Mario Botsch, Alexander Hornung, Matthias Zwicker, and Leif Kobbelt. 2005. High-quality surface splatting on today's GPUs. In *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005.* 17–141.

Christian Boucheny. 2009. *Interactive Scientific Visualization of Large Datasets: Towards a Perceptive-Based Approach.* Ph.D. Dissertation. Université Joseph Fourier, Grenoble.

Alexandre Boulch, Bertrand Le Saux, and Nicolas Audebert. 2017. Unstructured point cloud semantic labeling using deep segmentation networks. In *Eurographics Workshop on 3D Object Retrieval*, Vol. 2. 1.

Howard Butler, David C Finnegan, Peter J Gadomski, and Uday K Verma. 2014. plas.io: Open Source, Browser-based WebGL Point Cloud Visualization. In *AGU Fall Meeting Abstracts*.

Dong Chen, Ruisheng Wang, and Jiju Peethambaran. 2017. Topologically aware building rooftop reconstruction from airborne laser scanning point clouds. *IEEE Transactions on Geoscience and Remote Sensing* 55, 12 (2017), 7032–7052.

Martin Christen and Stephan Nebiker. 2015. Visualisation of complex 3D city models on mobile webbrowsers using cloud-based image provisioning. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 2 (2015), 517–522.

Rémi Cura, Julien Perret, and Nicolas Paparoditis. 2017. A scalable and multi-purpose point cloud server (PCS) for easier and faster point cloud data management and processing. *ISPRS Journal of Photogrammetry and Remote Sensing* 127 (2017), 39–56.

David Deibe, Margarita Amor, and Ramón Doallo. 2019. Supporting multi-resolution out-of-core rendering of massive LiDAR point clouds through non-redundant data structures. *International Journal of Geographical Information Science* 33, 3 (2019), 593–617.

David Deibe, Margarita Amor, Ramón Doallo, David Miranda, and Miguel Cordero. 2017. GVLiDAR: an interactive web-based visualization framework to support geospatial measures on lidar data. *International journal of remote sensing* 38, 3 (2017), 827–849.

Jürgen Döllner, Benjamin Hagedorn, and Jan Klimke. 2012. Server-based rendering of large 3D scenes for mobile devices using G-buffer cube maps. In *Proceedings of the 17th International Conference on 3D Web Technology.* 97–100.

Jan UH Eitel, Bernhard Höfle, Lee A Vierling, Antonio Abellán, Gregory P Asner, Jeffrey S Deems, Craig L Glennie, Philip C Joerg, Adam L LeWinter, Troy S Magney, et al. 2016. Beyond 3-D: The new spectrum of lidar applications for earth and ecological sciences. *Remote Sensing of Environment* 186 (2016), 372–392.

Jan Elseberg, Dorit Borrmann, and Andreas Nüchter. 2013. One billion points in the cloud–an octree for efficient processing of 3D laser scans. *ISPRS Journal of Photogrammetry and Remote Sensing* 76 (2013), 76–88.

Zhenzhen Gao, Luciano Nocera, Miao Wang, and Ulrich Neumann. 2014. Visualizing aerial LiDAR cities with hierarchical hybrid point-polygon structures. In *Proceedings of Graphics Interface 2014.* 137–144.

Sergio García, Rafael Pagés, Daniel Berjón, and Francisco Morán. 2015. Textured splat-based point clouds for rendering in handheld devices. In *Proceedings of the 20th International Conference on 3D Web Technology.* 227–230.

Prashant Goswami, Fatih Erol, Rahul Mukhi, Renato Pajarola, and Enrico Gobbetti. 2013. An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees. *The Visual Computer* 29, 1 (2013), 69–83.

Markus Gross and Hanspeter Pfister. 2011. *Point-based graphics.* Morgan Kaufmann.

Ralf Gutbell, Lars Pandikow, Volker Coors, and Yasmina Kammeyer. 2016. A framework for server side rendering using OGC's 3D portrayal service. In *Proceedings of the 21st International Conference on Web3D Technology.* 137–146.

Benjamin Hagedorn, Simon Thum, Thorsten Reitz, Volker Coors, and Ralf Gutbell. 2017. *OGC 3D Portrayal Service 1.0.* OGC Implementation Standard 1.0. Open Geospatial Consortium.

Jing Huang and Suya You. 2016. Point cloud labeling using 3d convolutional neural network. In *Proceedings of the 23rd International Conference on Pattern Recognition.* 2670–2675.

Andreas Jochem, Bernhard Höfle, Volker Wichmann, Martin Rutzinger, and Alexander Zipf. 2012. Area-wide roof plane segmentation in airborne LiDAR point clouds. *Computers, Environment and Urban Systems* 36, 1 (2012), 54–64.

Thomas P Kersten, Heinz-Jürgen Przybilla, Maren Lindstaedt, Felix Tschirschwitz, and Martin Misgaiski-Hass. 2016. Comparative geometrical investigations of hand-held scanning systems. *ISPRS Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* (2016).

Tobias Langner, Daniel Seifert, Bennet Fischer, Daniel Goehring, Tinosch Ganjineh, and Raúl Rojas. 2016. Traffic awareness driver assistance based on stereovision, eye-tracking, and head-up display. In *Proceedings of ICRA 2016.* 3167–3173.

Oscar Martinez-Rubi, Stefan Verhoeven, Maarten Van Meersbergen, M Schûtz, Peter Van Oosterom, Romulo Gonçalves, and Theo Tijssen. 2015. Taming the beast: Free and open-source massive point cloud web visualization. In *Proceedings of the Capturing Reality Forum 2015.*

Martin Mittring. 2007. Finding next gen: Cryengine 2. In *ACM SIGGRAPH 2007 courses.* ACM, 97–121.

Matthias Müller and Benjamin Pross. 2015. OGC WPS 2.0 interface standard. *Open Geospatial Consortium Inc.* (2015).

Stephan Nebiker, Susanne Bleisch, and Martin Christen. 2010. Rich point clouds in virtual globes–A new paradigm in city modeling? *Computers, Environment and Urban Systems* 34, 6 (2010), 508–517.

Steve Ostrowski, Grzegorz Jóźków, Charles Toth, and Benjamin Vander Jagt. 2014. Analysis of point cloud generation from UAS images. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 2, 1 (2014), 45–51.

Viorica Pătrăucean, Iro Armeni, Mohammad Nahangi, Jamie Yeung, Ioannis Brilakis, and Carl Haas. 2015. State of research in automatic as-built modelling. *Advanced Engineering Informatics* 29, 2 (2015), 162–171.

Shmuel Peleg, Moshe Ben-Ezra, and Yael Pritch. 2001. Omnistereo: Panoramic stereo imaging. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23, 3 (2001), 279–290.

Ruggero Pintus, Enrico Gobbetti, and Marco Agus. 2011. Real-time Rendering of Massive Unstructured Raw Point Clouds Using Screen-space Operators. In *Proceedings of VAST 2011.* 105–112.

Florent Poux, Pierre Hallot, Romain Neuville, and Roland Billen. 2016. Smart point cloud: Definition and remaining challenges. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 4 (2016), 119–127.

Reinhold Preiner, Stefan Jeschke, and Michael Wimmer. 2012. Auto Splats: Dynamic Point Cloud Visualization on the GPU.. In *Proceedings of the EGPGV.* 139–148.

Fabio Remondino, Maria Grazia Spera, Erica Nocerino, Fabio Menna, Francesco Nex, and Sara Gonizzi-Barsanti. 2013. Dense image matching: comparisons and analyses. In *Proceedings of DigitalHeritage 2013*, Vol. 1. 47–54.

Rico Richter, Markus Behrens, and Jürgen Döllner. 2013a. Object class segmentation of massive 3D point clouds of urban areas using point cloud topology. *International Journal of Remote Sensing* 34, 23 (2013), 8408–8424.

Rico Richter, Sören Discher, and Jürgen Döllner. 2015. Out-of-core visualization of classified 3d point clouds. In *3D Geoinformation Science.* Springer, 227–242.

Rico Richter, Jan E Kyprianidis, and Jürgen Döllner. 2013b. Out-of-Core GPU-based Change Detection in Massive 3D Point Clouds. *Transactions in GIS* 17, 5 (2013), 724–741.

Marcos B Rodriguez, Enrico Gobbetti, Fabio Marton, Ruggero Pintus, Giovanni Pintore, and Alex Tinti. 2012. Interactive Exploration of Gigantic Point Clouds on Mobile Devices.. In *13th International Conference on Virtual Reality, Archaeology and Cultural Heritage.* 57–64.

Szymon Rusinkiewicz and Marc Levoy. 2000. QSplat: A multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques.* 343–352.

Heinz Rüther, Christoph Held, Roshan Bhurtha, Ralph Schroeder, and Stephen Wessels. 2012. From point cloud to textured model, the zamani laser scanning pipeline in heritage documentation. *South African Journal of Geomatics* 1, 1 (2012), 44–59.

Takafumi Saito and Tokiichiro Takahashi. 1990. Comprehensible rendering of 3-D shapes. In *ACM SIGGRAPH Computer Graphics*, Vol. 24. ACM, 197–206.

Claus Scheiblauer and Michael Wimmer. 2011. Out-of-core selection and editing of huge point clouds. *Computers & Graphics* 35, 2 (2011), 342–351.

Markus Schütz and Michael Wimmer. 2015a. High-quality point-based rendering using fast single-pass interpolation. In *Proceedings of Digital Heritage 2015.* 369–372.

Markus Schütz and Michael Wimmer. 2015b. Rendering large point clouds in web browsers. *Proceedings of CESCG* (2015), 83–90.

Lance Simons, Stewart He, Peter Tittman, and Nina Amenta. 2014. Point-based rendering of forest LiDAR. In *Workshop on Visualisation in Environmental Sciences (EnvirVis), The Eurographics Association.* 19–23.

Tee-Ann Teo and Chi-Min Chiu. 2015. Pole-like road object detection from mobile lidar system using a coarse-to-fine approach. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 8, 10 (2015), 4805–4818.

Peter van Oosterom, Oscar Martinez-Rubi, Theo Tijssen, and Romulo Gonçalves. 2017. Realistic benchmarks for point cloud data management systems. In *Advances in 3D Geoinformation.* Springer, 1–30.

Michael Wand, Alexander Berner, Martin Bokeloh, Philipp Jenke, Arno Fleck, Mark Hoffmann, Benjamin Maier, Dirk Staneker, Andreas Schilling, and Hans-Peter Seidel. 2008. Processing and interactive editing of huge point clouds from 3D scanners. *Computers & Graphics* 32, 2 (2008), 204–220.

Long Zhang, Qian Sun, and Ying He. 2014. Splatting lines: an efficient method for illustrating 3D surfaces and volumes. In *Proceedings of the 18th meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games.* 135–142.